REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE Dec 1, 1994	3. REPORT TYPE AND DATES COVERED Final 15 Jul 92 - 15 Jul 95

4. TITLE AND SUBTITLE
ADA in Introductory Courses

5. FUNDING NUMBERS
DAAL03-92-G-0415

6. AUTHOR(S)
Dr. Sara Stoecklin
Dr. Marion Harmon

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Dr. Sara Stoecklin
Florida A&M University
Tallahassee, Florida 32307-2001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 30999.1-MA

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This project at Florida A & M University sponsored development of a new sequence of two lecture courses entitled (1) Fundamentals of Programming with Ada and (2) Program, File and Data Structures with Ada with two innovative supporting laboratory courses. These four courses replaced a sequence of courses previously taught with Pascal. The laboratory courses were developed to support the theory that reading programs and experimenting with those programs prior to writing original programs would be helpful in teaching complex programming languages such as Ada. The intent of this project was to allow students to learn a usable language during their first courses in programming and for these same students to develop projects using the ADA language during project courses. Additionally, there are several industries who needed programmers in ADA and these students were targeted for those types of industries.

SUBJECT TERMS
ADA, Introductory Courses, Programming Laboratory

15. NUMBER OF PAGES 32

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

ADA in Introductory Courses

FINAL REPORT

Dr. Sara Stoecklin
Dr. Marion Harmon

12/01/94

Accesion For	
NTIS CRA&I	[✓]
DTIC TAB	☐
Unannounced	☐
Justification ________	
By ________	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

U.S. Army Research Office

BAA #92 – 25
ARO 30999 – MA
DAAL03 – 92 – G – 0415

**FLORIDA A & M UNIVERSITY
HBCU**

PROGRESS REPORT

1. ARO PROPOSAL NUMBER: 300999–MA

2. PERIOD COVERED BY REPORT: 1 January 1993 – November 24, 1994

3. TITLE OF PROPOSAL: ADA in Introductory Courses

4. CONTRACT OR GRANT NUMBER: DAAL03–92–G–0415

5. NAME OF INSTITUTION: Florida A & M University

6. AUTHORS OF REPORT: Dr. Sara Stoecklin and Dr. Marion Harmon

7. LIST OF MANUSCRIPTS SUBMITTED OR PUBLISHED UNDER ARO SPONSORSHIP DURING THIS REPORTING PERIOD, INCLUDING JOURNAL REFERENCES:

   1. Stoecklin, S., and Harmon, M., Using Ada, 1993. (Text Publication for the Laboratory Course supporting the Introductory Ada Courses).
   2. Stoecklin, S., and Snead, D., Using Ada Programming Diskette, 1992 (Supporting Diskette with Text)
   3. Harmon, M. Course Material including Syllabus, Outlines, Course Notes, Assignments, Tests, and Quizzes

8. SCIENTIFIC PERSONNEL SUPPORTED BY THIS PROJECT AND DEGREES AWARDED DURING THIS REPORTING PERIOD:

Name	Degrees Awarded
1. Debra Snead	BS – CIS/FAMU
2. Tomeka Williams	BS – CIS/FAMU
3. Mary Clark	
4. Dr. Sara Stoecklin	
5. Dr. Marion Harmon	
6. Dr. Usha Chandra	

9. REPORT OF INVENTIONS (BY TITLE ONLY): (NONE)

Dr. Sara Stoecklin
Florida A&M University
Tallahassee, Florida 32307–2001

I. FORWARD

This project at Florida A & M University sponsored development of a new sequence of two lecture courses entitled (1) Fundamentals of Programming with Ada and (2) Program, File and Data Structures with Ada with two innovative supporting laboratory courses. These four courses replaced a sequence of courses previously taught with Pascal. The laboratory courses were developed to support the theory that reading programs and experimenting with those programs prior to writing original programs would be helpful in teaching complex programming languages such as Ada. The intent of this project was to allow students to learn a usable language during their first courses in programming and for these same students to develop projects using the ADA language during project courses. Additionally, there are several industries in Florida who needed programmers in ADA and these students were targeted for those types of industries.

II. TABLE OF CONTENTS

III. LIST OF APPENDIXES, ILLUSTRATIONS AND TABLES

IV. BODY OF REPORT

A. STATEMENT OF THE PROBLEM STUDIED

The problem studies was to develop courses which would teach the ADA language as the first programming language course for incoming students. Many students have trouble with problem solving, problem decomposition, data typing and data structures while writing those first programs. The theory tested here was to teach reading and basic experimenting with code prior to code development. This technique was to aid in the understanding of program development.

B. SUMMARY OF THE MOST IMPORTANT RESULTS

1. Information and Technology Sharing
   The most significant result of the project was the sharing of information between universities trying various techniques with Ada. The results and laboratory course developed as part of this project was shared with two other universities trying to teach Ada as their first programming course. East Tennessee University, one of those universities, used the laboratory course material in the development of their own programming courses to support a software engineering effort.

2. Improved Learning through Laboratory Experiences
   A significant result of the project was the faculty noting that students certainly benefited from reading code prior to writing code. Currently, our first programming courses has seen so much use in these activities that this part of the course has become more important. The potential integration of the extra laboratory developed is now being considered to be a critical part of the actual introductory course.

3. Ada Environment Support
   Another significant result of the project was the development of support for the Ada environment. Students who worked in this environment were significantly better prepared for internships and work using the Ada programming language.

4. Ada Students
   The most valuable result of this project were the students who continue to be exposed to the ADA programming language and have added to the number of Ada trained personnel in the job market.

C. LIST OF ALL PUBLICATIONS AND TECHNICAL REPORTS

1. Stoecklin, S., and Harmon, M., Using Ada, 1993. (Text Publication for the Laboratory Course supporting the Introductory Ada Courses).
2. Stoecklin, S., and Snead, D., Using Ada Programming Diskette, 1992 (Supporting Diskette with Text)
3. Harmon, M. Course Material including Syllabus, Outlines, Course Notes, Assignments, Tests, and Quizzes

D. LIST OF ALL PARTICIPATING SCIENTIFIC PERSONNEL

Name	Degrees Awarded
1. Debra Snead	BS – CIS/FAMU
2. Tomeka Williams	BS – CIS/FAMU
3. Mary Clark	
4. Dr. Sara Stoecklin	
5. Dr. Marion Harmon	
6. Dr. Usha Chandra	

V. REPORT IF INVENTIONS (BY TITLE ONLY) – none

VI. BIBLIOGRAPHY

[BOO91] Booch, G., Object–Oriented Development, Benjamin Cummins, 1991.

[BRO87] Brooks, F., "No Silver Bullet", Computer, April 1987.

[DAL91] Dale, N., and Wiggins, R., "Computer Programming with Pascal, Heath, 1991.

[MIL89] Mills, H. "The Clean Room Approach," IEEE Software Engineering, 1989.

[MIL90] Miller, N.E. and Peterson, C.G., File Structures with Ada, Benjamin Cummins, 1990.

[PET85] Petroski, H., To Engineer is Human, St Martin's Press, New York, 1985.

[SAD92] Sadidch, W.J., and Peterson, C.G., Ada and Introduction to the Art and Science of Programming Benjamin Cummins, 1992.

VII. APPENDIXES

APPENDIXES

COP1215 SYLLABUS

Fundamentals of Programming with ADA

PrerequisiteMAC 1104 and MAC 1133 OR MAC 1142, OR CALC I
Co-requisite COP1215L AND MAD2102

OVERVIEW OF COURSE

This course is designed to prepare students for analyzing problems and for designing and implementing algorithms by understanding language concepts and the ADA programming language.

COURSE OBJECTIVES

1. Introduction to the programming progress; problem analysis, algorithm development, verification of algorithm, coding, debugging, testing, documentation, and maintenance.
2. Introduction to the top-down methodology and the use of structure charts and pseudocode to develop algorithms.
3. Introduction to the syntactic and semantic rules of ADA.
4. Introduction to the stylistic issues in coding of programs.

CRITICAL PREREQUISITE KNOWLEDGE AND SKILLS

1. Students should have a working knowledge of Boolean algebra operations.
2. Students should have a working knowledge of relational operators and mathematical operators.
3. Students should be able to apply problem solving techniques to solve algebraic problems.

PERFORMANCE OBJECTIVES FOR STUDENTS

Upon Completion of this course, students should be able to perform the following tasks:

1. Use the ADA Environment to create, edit, compile, and execute an ADA program.
2. Distinguish between computer hardware and software.
3. List the basic programming steps.
4. Apply the top-down methodology to solve simple to moderate problems.
5. Know the difference between batch and interactive processing.
6. Code an algorithm in ADA using self-documenting code.
7. Trace the execution of a program.
8. Select adequate data sets for testing of programs.
9. Develop algorithms which require the use of selection, repetition, and procedure control structures.
10. Given a design which requires sub-tasks,
    a. Determine what the formal and actual parameter list should be for each module.
    b. Determine the method of parameter passing, call by reference versus call by value.
    c. Determine scope of variables, local versus global
11. Manipulate an array of record data type and its operations.

CONCEPTUAL OBJECTIVES FOR STUDENTS

Upon the successful completion of this course, the student will understand:

1. The computer is an electronic device which only follows instructions.
2. There are several steps to the programming process.
3. One method of design is the top-down methodology.
4. Data types and their operations.
5. Design tools, structure charts, N-S charts, pseudocode
6. Techniques for problem solving
7. Data validation and testing strategies

PROPOSED SCHEDULE

WEEK	TOPIC	
1	Chapter 1	Overview of Programming and Problem Solving
2	Chapter 2	ADA Syntax and Semantics
3.	Chapter 3	Input and Design Methodology **First Program
4.	Chapter 4	Conditions, Boolean Expressions, and Selection Control Structures. **Second Program – Read format/use file
5.	Chapter 5	Looping
6.	Chapter 5/6	**Third Program – Selection/ Loops
7.	Chapter 6	Procedures
8.	Chapter 7	Value Parameters, Nesting, Procedures, and More on Interface Design **Fourth Program – Procedures
9.	Chapter 8	Functions, Precision, and Recursion
10.	Chapter 9	Sets and Additional Control Structures
11.	Chapter 10	Simple Data Types **Fifth Program – Procedures
12	Chapter 11	One – Dimensional Arrays
13	Chapter 12	Applied Arrays: Lists and Strings
14	Chapter 13	Multi – dimensional Arrays **Sixth Program – Arrays
15	Chapter 14	Records and Data Abstractions
16	Chapter 15	Exams

MIDTERM

ADA Programming
Language Exam

NAME____________________

DATE ____________________

(1 point each)
TRUE/FALSE.

_________1) The ada programming language was developed on behalf of the U.S. Department of Defense to help control the cost of software for computers embedded in larger systems.

_________2) The parameter mode OUT specifies that the parameter will be used to transmit information from the calling program or subprogram.

_________3) The following statement would be ignored by the ada compiler and assumed to be a comment:

--- The Program computes ....

________4) In ada identifiers, may consist of letter, digits, and underscores(_).

________5) When a new compulation unit uses entities defined in older, separately compiled units, the new unit must begin with a occurs clause naming the older units.

_______6) An Ada program is simply a subprogram with parameters.

_______7) In an ada package only those subprograms defined in the package specifications can be call from outside the package.

_______8) To give more precise control over entities provided to the outside world, the Ada language allows a package specification to contain a private part, which begins with the word PRIVATE and extends to the end of the Package specifications.

______9) Floating-point types are approximations of real numbers and are evenly spaced.

_____10) Declarations are processed one after the other at the time a program is compiled.

_____11) A discrete type is either an integer type or an enumeration type.

_____12) An allocator is a expression whose evaluation causes dynamic allocation of a variable.

_____13) An incomplete type declaration must be followed immediately by a full declaration for the same type.

_____14) The following FOR LOOP WILL PRINT "HELLO" 4 times?

```
type DAY_OF_WEEK_TYPE IS (MON, WED, THUR, FRI)

 FOR I IN   FRI .. MON LOOP
  TEXT_IO.PUT("HELLO");
 END LOOP;
```

_____15) Assume the following :

```
TYPE NUM_PTR is Access STRING(1..5);
ZIP_CODE_PTR : NUM_PTR;
BEGIN
  ZIP_CODE_PTR := NEW STRING;  is this statement legal?
    :
    :
    :
```

(2 POINTS EACH)
MULTIPLE CHOICE.

_______1) WHICH OF THE FOLLOWING IS NOT A RESERVED WORD IN ADA.

A) OUT B) DIGIT

C) LOOP D) DELTA E) ELSIF

_______2) Which of the following declarations is INVALID.

a) LIST: CONSTANT STRING(1..5) := "sting";

b) X , Y, Z : INTEGER := 0;

c) A : INTEGER := 4;

d) ch : CHARACTER range 'A' .. 'C';

e) none of these

_______3) Which of the following operators can be used to catenate to operands.

a) ^ b) +

c) / d) & e) none of these

______4) For what value of X will the following condition to evaluate to TRUE. ASSUME THAT Y IS 7.

```
NOT (X = 5) AND (Y > X) OR X < 10
```

A) 5 ONLY B) ANY NUMBER LESS THAN 10 C) 10
D) A NUMBER LESS THAN 7 BUT GREATER THAN 4

E) NONE OF THESE

______5) Which of the following forms of an allocator is INVALID.

a) NEW typemark b) NEW typemark ^(expression)

c) NEW typemark index-constraint

d) NEW typemark '(expression)

e) NEW typemark " (expression)

SHORT ANSWERS.

(5 points each)
1)
Write an ada subprogram to overload the "<" operator. This subprogram should be return the small of two integer operands. FOR example x:= 5 < 7; would store a 5 in x since 5 is less than 7. BUT for x := 5 < 4; x would get the value 4. Your subprogram should accept two integer parameters.

2) DECLARE A RECORD WITH THE FOLLOWING FIELDS.

```
  RECORD
     NAME: 20 characters
     age: 1..120
     eye color : blue, brown, green, gray, unknown
     batting average : 0.0 .. 1.0  to the nearest thousandth
                        precision.
     height, weight: Float 3 digits accuracy.
 end record:
```

3) Declare and unconstrained array of the records declared above.

4)
```
    TYPE DAY_TYPE is (MON, TUES, WED, THUR, FRI, SAT, SUN);

    X : DAY_TYPE;

    IF X = MON then
        TEXT_IO.PUT("MONDAY");
    ELSIF (X = FRI) and then (Y = SAT) then
        TEXT)IO.PUT("The Weekend is Near");
    ELSEIF (X = TUES) then
        TEXT_IO.PUT("TUESDAY");
    END IF;
```

REWRITE THE IF STATEMENT ABOVE USING A CASE STRUCTURE.

5) WRITE A FUNCTION TO COMPUTE THE FOLLOWING SUM. THE FUNCTION SHOULD RETURN THE SUM OF THE FOLLOWING SERIES.

2 + 4 + 8 + 16 + 32 + ... + 512 + 1024

(2 points each)
GIVEN the type declarations

```
TYPE ZIP_TYPE is RANGE 1 .. 99999.
TYPE SPEED_TYPE is DIGITS 6;
TYPE AVG_SPEED_TYPE is DELTA 2.5 RANGE 0.0 .. 200.0
TYPE ARRAY_TYPE is ARRAY (INTEGER RANGE <>) of CHARACTER;
TYPE COLOR_TYPE is (RED, BLUE, GREEN, PINK, ORANGE);
```

indicate which of the following object declarations have appropriate constraints and what is wrong with the declarations that do not have appropriate constraints.

a) table_of_char: ARRAY_TYPE(1 .. 50); ____________________

b) SPEED_1 : SPEED_TYPE DIGITS 7 RANGE 0.3 .. 10.0 __________

c) COLOR := COLOR_TYPE := COLOR_TYPE'LAST; _________________

d) SPEED_3 : AVG_SPEED_TYPE DELTA 1.5; ____________________

(2 points each)
EVALUATE:

a) -5 REM 2 ___________

b) -8 MOD -4 ___________

c) 6 REM 4 ___________

d) 26 REM -5 ___________

(5 points)

Write the follwoing sequence of statements without a GOTO statement.

```
  DiVISOR := 2;
  WHILE (DIVISOR ** 2 < = CHANDIDATE) LOOP
   IF (CANDIDATE MOD DIVISOR = 0) THEN
      GOTO not_prime;
   END IF;
  END LOOP;

 -- if this point is reached then the candiate must be prime

 << not_prime >>
   CANDIDATE := CANDIDATE + 1;
```

CONSIDER the following declarations:

```
TYPE HOUR_TYPE IS RANGE 1 .. 12;

SUBTYPE LATE_HOUR_SUBTYPE IS
        HOUR_TYPE RANGE 5 .. 10;

TYPE DAY_TYPE IS RANGE 1 .. 7;

SUBTYPE WORK_DAY_SUBTYPE IS DAY_TYPE RANGE 1 .. 6;

 HOUR_NUM        : HOUR_TYPE;

 LATE_HOUR_NUM   : LATE_HOUR_SUBTYPE;

 DAY_NUM         : CONSTANT DAY_TYPE;

 WORK_DAY        : WORK_DAY_SUBTYPE;
```

YOU MAY ASSUME THAT LEGAL ASSIGNMENTS HAVE BEEN MADE TO THESE OBJECTS AT SOME POINT DURING EXECUTION.

(2 points each)
Which of the following assignments are not legal and why?

a) HOUR_NUM := LATE_HOUR_NUM; ____________________

b) LATE_HOUR_NUM := HOUR_NUM; ____________________

c) DAY_NUM := 3; ________________________________

d) WORK_DAY := DAY_NUM ____________________________

QUIZ 1
COP4900 NAME________________________
DATE________________________

True or False.
(5 points)

_____ Ada was developed on behalf of the U. S. Department of Defense stricly for the development of embedded computer systems.

_____ Ada is a standarderdized high level programming language.

_____ Ada provides for separate compilation for separate subprogram components.

_____ In ada it is possible to declare more than one subprogram with the same name, if they do not have the same parameters and result types, this is called redefining names.

_____ A data type that can be completely characterized by the way in which the values of the data are related to each other by the operations.

_____ A package is a collection of related entities that can be used by multiple programs.

_____ A draws now distinction between the external appearance of a package and its internal workings.

_____ The following is not and ada reserved word WHEN.

_____ The following is an illegal ada identifier FIVE_%.

(5 points)
Locate the syntax errors in the following Ada Program:

```
With Basic_IO;
PROCEDURE EXAMPLE is
J, L : INTEGER
BEGIN
FOR I in 1..5 loop
  PUT("ENTER A NUMBER ');
  BASIC_IO.GET(J);
  NEW_LINE;
  PUT("ENTER ANOTHER NUMBER ");
  BASIC_IO.GET(L);
  NEW_LINE;
  J := J + L:
  PUT("THE RESULT IS ");
  NEW_LINE;
  PUT(J)
end
END EXAMPLE;
```

Number the instances of syntax errors and explain why be side the number:

1) ______________________________________

2) ______________________________________

3) ______________________________________ there are more than 3

(10 points)
Write an ada program that accepts a sentence from the screen and prints the number of vowels in the sentence.

example:

The cat chased the mouse across the room.

number of vowels is 13.

The program will be graded on correctness and style.

THE COLOR PALATE ASSIGNMENT

You are to write an Ada program which will mix colors and display their resultant color. In order to accomplish this task, you will probably need to use the following Ada concepts:

Overloading of an operator (use '+')
Enumeration of the colors (declare a type)
Enumeration I/O (instantiate enumeration I/O)
Loop, Case, If constructs
Put/Get to/from the terminal (implies use of TEXT_IO)

Your program should use the following color palate as a minimum, but you are free to add other colors and define their resultant colors as you deem appropriate.

First Color	Second Color	Resultant Color
RED	YELLOW	ORANGE
BLUE	RED	PURPLE
BLUE	YELLOW	GREEN

Don't forget to consider the inverse cases, e.g., if YELLOW is the first color and RED is the second color, then their resultant mix is still ORANGE. Any other combination is not defined, unless you choose to do so.

Your program should follow the following scenario. It should prompt the user for the first color, then read that color. Next, it should prompt for the second color and read that color. Then it should call a function which determines what the mix will be if there is one. (You should strongly consider overloading the '+' operator as your function call.) The resultant color, if there is one, should then be printed; otherwise, a message should be printed telling the user that the mix is undefined. Finally, you should prompt the user to determine if the user desires to mix other colors. If so, then loop back through this process; if not, then exit the loop.

GOOD LUCK!

Unit 2

Introduction to the Ada Language

Conceptual Objectives

1. Understand the Ada Compiler
2. Understand the Basic Commands of the Operating System
3. Understand the Basic Commands of the Compiler
4. Understand the Basic Commands of the Editor
5. Understand How to Run an Ada Program
6. Recognize Overloading

Performance Objectives

1. Be able to use DOS commands such as
   COPY, ERASE, TYPE, DIR, and PRINT for files
2. Be able to RETRIEVE, STORE, and EDIT an Ada Program
3. Be able to COMPILE and RUN an Ada Program
4. Be able to Traverse through the Ada Program
5. Be able to Run an Ada Program using Different Data Types

Ada Programs

The Ada programs used in this chapter are composed of two types of statements, declaration statements and execution statements. Declaration statements used in this chapter include the procedure statement, identifier statement, and the included statements of other packages. The execution statements used in this chapter are the assignment statement and the output statements. A short description is included below with examples of each of the statements.

Procedure Statement

Each Ada program is given a program name by the procedure statement. This name is used to link and compile the program. In addition, the program is stored in the DOS directory under a filename. These two names do not have to be the same. The procedure statement has the keyword **procedure**, the name of a valid identifier in the Ada language, and the keyword **is**. This statement is followed by the Ada statements necessary for the program and the end end statement for the procedure. The syntax of the procedure statement to declare the program name is:

```
procedure IDENTIFIER is
begin
   -- Ada statements go here
end IDENTIFIER;
```

Declaration Statements – Identifiers

There are various declarations which are made through valid Ada identifiers. Such things as Variables and Constants are described using simple data types such as integer, float(real), fixed, character and Boolean. The declaration statements contain the name of the identifier and the data type for that identifier. The syntax of the variable statements to declare these identifiers to the Ada program is:

```
IDENTIFIER : INTEGER;
IDENTIFIER : FLOAT;
IDENTIFIER : CHARACTER;
```

The syntax of the constant statement is:

```
IDENTIFIER : CONSTANT INTEGER := 10;
IDENTIFIER : CONSTANT FLOAT := 10.0;
IDENTIFIER : CONSTANT CHARACTER := "A";
```

I/O Packages

During the first few weeks a simple input/output (I/O) package will be used to allow your Ada program to input values from the keyboard and output values to the screen. This package is called

Simple_Ada_IO

Output Statements

There are two statements in the Simple_Ada_IO that allow the user to express information to an output device, such as the screen. These two statements are **PUT and PUT_LINE**. The first statement, **PUT**, will display on the output device with no advancement to the next line (i.e. no carriage return [CR] and line feed [LF]). The second statement, **PUT_LINE**, will display on the output device with an advancement in the line. Variables, constants, and literals are output using these two statements. Literals are strings of characters such as those characters on the keyboard. Variables and constants are those identifiers declared by the declaration statements. The syntax of expressing a variable, constant, and literal on an output device are:

```
SIMPLE_ADA_IO.PUT ( IDENTIFIER );      <-- displays identifier's value
SIMPLE_ADA_IO.PUT_LINE ( "ABC" );      <-- displays the literal  ABC.
```

Assignment Statement

The assignment statement allows value assignments to occur to an identifier on the left hand side of the :=. If there is an identifier on the right hand side of the := such as IDENTIFIER2 then the value of IDENTIFIER2 is **assigned** to the the identifier on the left hand side of the := such as IDENTIFIER1. If there is an arithmetic expression on the right side of the := then the value of the expression is calculated and assigned to the identifier on the left hand side of the := such as IDENTIFIERA. The syntax of the assignment statement is:

```
IDENTIFIER1 := IDENTIFIER2;
IDENTIFIERA := arithmetic expression;
```

Use the program. named Temp_Conversion, printed below for the following laboratories. The program is located in the file called UNIT2.A1. Review the temperature conversion program to understand how it converts a temperature from Fahrenheit to Celsius and from Celsius to Fahrenheit.

NOTE: The two dashes -- located together identify comment statements embedded in the Ada program and have NO relevance to the execution of the program. They DO serve as documentation to the program to allow programmers to make comments within the program for better program understanding.

```
==========================================================

-- UNIT2.A1

with SIMPLE_ADA_IO;

procedure TEMP_CONVERSION is

-- This programs allows the user to enter a temperature.
-- The temperature can be either Fahrenheit or Celsius.
-- A temperature in Fahrenheit is converted to Celsius.
-- A temperature in Celsius is converted to Fahrenheit.

  TEMP_IN_FAHRENHEIT : constant INTEGER := 32;
  TEMP_IN_CELSIUS : constant INTEGER := 0;
  FAHRENHEIT_TO_CELSIUS : INTEGER;     -- RESULT
  CELSIUS_TO_FAHRENHEIT : INTEGER;     -- RESULT

begin   -- TEMP_CONVERSION
   CELSIUS_TO_FAHRENHEIT := (9*TEMP_IN_CELSIUS + 160)/5;
   FAHRENHEIT_TO_CELSIUS := 5*(TEMP_IN_FAHRENHEIT - 32)/9;
   SIMPLE_ADA_IO.PUT (TEMP_IN_FAHRENHEIT);
   SIMPLE_ADA_IO.PUT (" in Fahrenheit is ");
   SIMPLE_ADA_IO.PUT (FAHRENHEIT_TO_CELSIUS);
   SIMPLE_ADA_IO.PUT (" in Celsius.");
end TEMP_CONVERSION;

==========================================================
```

LABORATORY 1

The program named TEMP_CONVERSION is located in a file named UNIT2.A1.

1. Get the file named UNIT2.A1 into the ACE editor.

2. Edit the program
   Change the last PUT statement to a PUT_LINE statement.
   Assure yourself that you made the change correctly.

2. Compile and Link the Program.
   IF THE PROGRAM DOES NOT COMPILE then the statement you inserted must be incorrect.

3. Run the program (execute the program).

   What temperature was calculated for Celsius? ________

Laboratory 1 completed ________

LABORATORY 2:

The program computes values for each of the formulas. One value is Celsius to Fahrenheit and the other is Fahrenheit to Celsius. There is a PUT statement for only one of these calculations of Fahrenheit to Celsius.

1. Edit the Program.
   Add a PUT_LINE statement to PUT the other calculations.
   Assure yourself that you made the change correctly.

2. Compile and Link and Run the program.

   What temperature was calculated for Celsius? ________

   What temperature was calculated for Fahrenheit? ________

Laboratory 2 completed ________

LABORATORY 3

Two constants are declared in the Ada program TEMP_CONVERSION.

1. Edit the Program.
   Change the value of the constant TEMP_IN_FAHRENHEIT to 212.
   Change the value of the constant TEMP_IN_CELSIUS to 100.

2. Compile and Link your program.

3. Run your program.
   What is the output for Fahrenheit to Celsius? ________

   What is the output for Celsius to Fahrenheit? ________

Laboratory 3 completed ________

LABORATORY 4

Parentheses in the equation set mathematical precedence of the calculations. Deleting or changing them can make a difference in the mathematical calculations.

1. Edit your program.
   Change the constant value TEMP_IN_FAHRENHEIT back to 32.
   Change the constant value TEMP_IN_CELSIUS back to 0.

2. Compile, Link, and Run your program.
   What is the output for Fahrenheit to Celsius? ________

   What is the output for Celsius to Fahrenheit? ________

3. Edit your program
   Delete the parentheses from the first assignment statements.

4. Compile, Link, and Rerun you program.

   What is the output for Fahrenheit to Celsius? ________

   What is the output for Celsius to Fahrenheit? ________

   Is the output the same as in question 3? Why or Why not?

Laboratory 4 completed ________

Program UNIT2_EXERCISE (UNIT2.A2) is a segment of a program. A segment is an outline of a program with sections of the program intentionally omitted. Use this program segment for Laboratories 5 and 6.

```
================================================================

 -- UNIT2.A2

       -- PLACE YOUR CODE HERE TO ACCESS Simple_Ada_IO.

 procedure UNIT2_EXERCISE is

 begin -- UNIT2_EXERCISE

       -- PLACE YOUR CODE HERE

 end UNIT2_EXERCISE;

================================================================
```

LABORATORY 5

Write a program to display the following information on the screen. Use literal constants to identify each of the data items written on the screen.

a. Your last name

b. Your birth date

Laboratory 5 completed ________

LABORATORY 6

Now change your solution for laboratory 5 to use named constants for month, day, and year rather than the literal constants. You will need a constant definition in the declaration section to your Ada program.

Laboratory 6 completed ________

SIMPLEIO.ADA

You will note that each of the programs use a package SIMPLE_ADA_IO. This program, shown below, allows Get and Put data to the screen and keyboard.

```
-- SIMPLEIO.ADA with IO_EXCEPTIONS; with TEXT_IO;
------------------------------------------------------------------
package SIMPLE_ADA_IO IS
   procedure GET (ITEM : out INTEGER);
   procedure GET (ITEM : out CHARACTER);
   procedure GET (ITEM : out STRING);
   procedure GET (ITEM : out FLOAT);
   procedure GET (FILE : in TEXT_IO.FILE_TYPE; ITEM : out INTEGER);
   procedure GET (FILE : in TEXT_IO.FILE_TYPE; ITEM : out CHARACTER);
   procedure GET (FILE : in TEXT_IO.FILE_TYPE; ITEM : out STRING);
   procedure GET (FILE : in TEXT_IO.FILE_TYPE; ITEM : out FLOAT);
   procedure PUT (ITEM : in INTEGER);
   procedure PUT (ITEM : in CHARACTER);
   procedure PUT (ITEM : in STRING);
   procedure PUT (ITEM : in FLOAT);
   procedure PUT (FILE : in TEXT_IO.FILE_TYPE; ITEM : in INTEGER);
   procedure PUT (FILE : in TEXT_IO.FILE_TYPE; ITEM : in CHARACTER);
   procedure PUT (FILE : in TEXT_IO.FILE_TYPE; ITEM : in STRING);
   procedure PUT (FILE : in TEXT_IO.FILE_TYPE; ITEM : in FLOAT);
   procedure GET_LINE (ITEM : out STRING;      LAST : out NATURAL);
   procedure PUT_LINE (ITEM : in STRING);
   procedure NEW_LINE;
   procedure NEW_PAGE;
   function END_OF_FILE return BOOLEAN;
   function END_OF_LINE return BOOLEAN;
   DEVICE_ERROR : EXCEPTION renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR    : EXCEPTION renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR   : EXCEPTION renames IO_EXCEPTIONS.DATA_ERROR;
end SIMPLE_ADA_IO;
------------------------------------------------------------------
package body SIMPLE_ADA_IO is
   package TYPE_INTEGER_IO is new TEXT_IO.INTEGER_IO (INTEGER);
   package TYPE_FLOAT_IO   is new TEXT_IO.FLOAT_IO (FLOAT);
   procedure GET (ITEM : out INTEGER) is
         begin          TYPE_INTEGER_IO.GET (ITEM);                   end GET;
   procedure GET (ITEM : out CHARACTER ) is
         begin          TEXT_IO.GET (ITEM);                           end GET;
   procedure GET (ITEM : out FLOAT) is
         begin          TYPE_FLOAT_IO.GET (ITEM);                     end GET;
   procedure GET (ITEM : out STRING) is
         begin          TEXT_IO.GET (ITEM);                                   end GET;
   procedure GET (FILE : in TEXT_IO.FILE_TYPE; ITEM : out INTEGER) is
          begin         TYPE_INTEGER_IO.GET (FILE, ITEM);     end GET;
   procedure GET (FILE : in TEXT_IO.FILE_TYPE; ITEM : out CHARACTER) is
         begin          TEXT_IO.GET (FILE, ITEM);                     end GET;
   procedure GET (FILE : in TEXT_IO.FILE_TYPE; ITEM : out FLOAT) is
         begin          TYPE_FLOAT_IO.GET (FILE, ITEM);               end GET;
   procedure GET (FILE : in TEXT_IO.FILE_TYPE; ITEM : out STRING) is
         begin          TEXT_IO.GET (FILE, ITEM);                     end GET;
   procedure PUT (ITEM : in INTEGER) is
         begin          TYPE_INTEGER_IO.PUT (ITEM);                   end PUT;
   procedure PUT (ITEM : in CHARACTER) is
         begin          TEXT_IO.PUT (ITEM);                                   end PUT;
   procedure PUT (ITEM : in FLOAT) is
         begin          TYPE_FLOAT_IO.PUT (ITEM);                     end PUT;
   procedure PUT (ITEM : in STRING) is
         begin          TEXT_IO.PUT (ITEM);                                   end PUT;
   procedure PUT (FILE : in TEXT_IO.FILE_TYPE; ITEM : in INTEGER) is
         begin          TYPE_INTEGER_IO.PUT (FILE, ITEM);     end PUT;
   procedure PUT (FILE : in TEXT_IO.FILE_TYPE; ITEM : in CHARACTER) is
         begin          TEXT_IO.PUT (FILE, ITEM);                     end PUT;
   procedure PUT (FILE : in TEXT_IO.FILE_TYPE; ITEM : in FLOAT) is
         begin          TYPE_FLOAT_IO.PUT (FILE, ITEM);               end PUT;
   procedure PUT (FILE : in TEXT_IO.FILE_TYPE; ITEM : in STRING) is
         begin          TEXT_IO.PUT (FILE, ITEM);                     end PUT;
   procedure GET_LINE (ITEM : out STRING;  LAST : out NATURAL) is
         begin          TEXT_IO.GET_LINE (ITEM, LAST);                        end GET_LINE;
   procedure PUT_LINE (ITEM : in STRING) is
         begin          TEXT_IO.PUT_LINE (ITEM);                              end PUT_LINE;
   procedure NEW_LINE is
         begin          TEXT_IO.NEW_LINE;                             end NEW_LINE;
   procedure NEW_PAGE is
         begin          TEXT_IO.NEW_PAGE;                                     end NEW_PAGE;
   function END_OF_FILE return BOOLEAN is
         begin          return TEXT_IO.END_OF_FILE;                   end END_OF_FILE;
   function END_OF_LINE return BOOLEAN is
         begin          return TEXT_IO.END_OF_LINE;                   end END_OF_LINE;
   end SIMPLE_ADA_IO;
------------------------------------------------------------------------
```